

This Page Is Inserted by IFW Operations  
and is not a part of the Official Record

## **BEST AVAILABLE IMAGES**

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

## **IMAGES ARE BEST AVAILABLE COPY.**

As rescanning documents *will not* correct images,  
Please do not report the images to the  
Image Problem Mailbox.

**This Page Blank (uspio)**



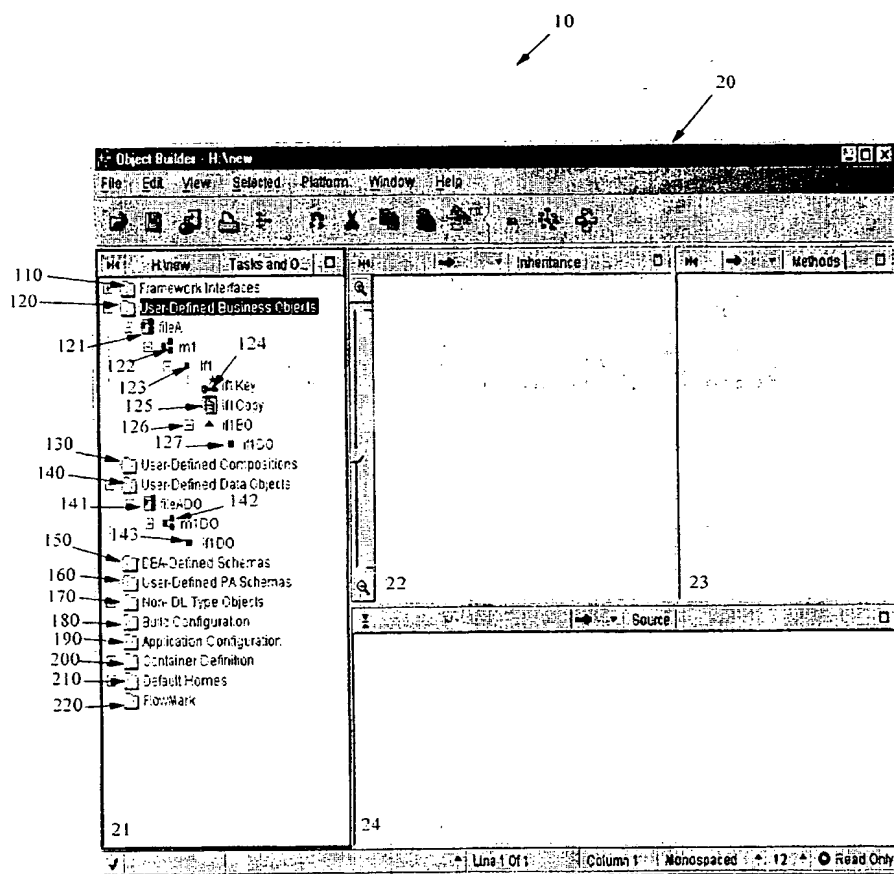
(72) LAU, Christina P., CA

(71) IBM CANADA LTD. - IBM CANADA LIMITÉE, CA

(51) Int.Cl.<sup>6</sup> G06F 9/44, G06F 9/46, G06F 17/30, G06F 3/14

(54) **METHODE ET MECANISME RELATIFS A UN MODELE DE  
DONNEES XML ORIENTE TACHES**

(54) **METHOD AND MECHANISM FOR A TASK ORIENTED XML  
DATA MODEL**



(57) A task oriented data model for an object oriented development tool. The data model comprises a task oriented structure which mirrors the task tree embodied in the user interface of the object development tool. The object development tool exports the data model as a document expressed in meta data language such as XML. The XML data model document comprises a series of data elements which are arranged according to a meta data model to mirror the user interface task tree. The meta data model is implemented in XML as a Document Type Definition and comprises containment relationships defined by XML constructs. The task oriented data model provides granularity for exporting and importing data elements, and also facilitates locating data elements of interest in the data model.



## METHOD AND MECHANISM FOR A TASK ORIENTED XML DATA MODEL

5                    ABSTRACT OF THE DISCLOSURE

A task oriented data model for an object oriented development tool. The data model comprises a task oriented structure which mirrors the task tree embodied in the user interface of the object development tool. The object development tool exports the data model as a document expressed in meta data language such as XML. The XML data model document comprises a series of data elements which are arranged according to a meta data model to mirror the user interface task tree. The meta data model is implemented in XML as a Document Type Definition and comprises containment relationships defined by XML constructs. The task oriented data model provides granularity for exporting and importing data elements, and also facilitates locating data elements of interest in the data model.

**TITLE:     METHOD AND MECHANISM FOR A TASK ORIENTED XML  
              DATA MODEL**

5       A portion of the disclosure of this patent document  
contains material which is subject to copyright  
protection. The copyright owner has no objection to the  
facsimile reproduction by anyone of the patent document  
or the patent disclosure, as it appears in the Patent  
10      Office patent files or records, but otherwise reserves  
all copyrights whatsoever.

**REFERENCE TO RELATED APPLICATIONS**

15      This application is related to the co-pending application  
filed in the name of the common assignee and entitled Method and  
Mechanism for a Task Oriented Data Model.

**FIELD OF THE INVENTION**

20      The present invention relates to object oriented programming  
systems, and more particularly to a task oriented structure for a  
data model.

**BACKGROUND OF THE INVENTION**

25      Traditionally in the computer programming arts, program code  
and data have been kept separate. For example, in the well-known  
C programming language, units of program code are called functions,  
and units of data are called structures. In C the functions and  
structures are not formally connected, that is, a function can  
operate on more than one type of data structure, and more than one  
30      function can operate on the same structure.

Object-oriented programming, in contrast, deals with objects,  
in which program code and data are merged into a single indivisible  
entity or "object". An object is an instance of an implementation

and an interface, which models a real-world entity. The object is implemented as a computational entity that encapsulates state and operations and responds to requests. The data and methods are implemented internally in the object. A business object for example is an object containing business methods (i.e. logic) and state that is intended for use within business applications.

Objects are built using an application program commonly referred to as an object builder tool. The object builder is a program which is used to define the business objects and generate the necessary definition and implementation files in IDL (Interface Definition Language), C++ and/or Java, as will be understood by those skilled in the programming arts.

Object builders utilize a data model which provides a template of the data structure for the objects. Object-oriented data models can be adapted to a wide range of different models by changing the definitions of meta-data, such as the objects and the task trees. Meta-data as will be understood by those skilled in the art comprises the self-descriptive information that can describe both services and information. With meta-data new services can be added to a system and discovered at run-time.

The object builder in the Component Broker™ toolkit from IBM utilizes a proprietary data model known as the Common Data Model (CDM), which is implemented as a binary file. In the object builder tool for Component Broker™, objects are processed by performing sets of user-defined operations in the form of task trees.

The binary file format of the Common Data Model yields some advantages such as rapid saving and loading of the data model. However, these advantages are offset by a number of disadvantages. First, the binary format of the Common Data Model or CDM is not human readable and as such the data model can only be examined by

using the user interface in the object builder or by viewing the code generated from the binary file. Second, the data model must be imported or exported in its entirety, i.e. component parts of the data model cannot be imported or exported. This makes it much more time-consuming to perform impact analysis and to compare different model-generated outcomes. For example, to compare the modelling results obtained by modifying one object, it is usually necessary to import or export the entire data model, even though the data model is substantially unchanged in any respect other than the slightly altered object.

This problem has been addressed in other data models by using a proprietary language, for example, the known Rational Rose tool utilizes a Rose mdl format file to represent the meta-data for the data model. Because the mdl format has been well publicized, objects using the mdl format can be designed and analyzed by any modelling tool which supports mdl. Using a proprietary language such as Rose mdl, the meta-data is organized so that the component parts of the data model, such as objects and task trees, can be imported and exported at various different levels of generality. However, using a proprietary language limits the inter-operability between the data model and other tools as these other tools will need to use parsers that are compatible with the proprietary language in order to retrieve meta-data from the data model.

Accordingly, there remains a need for generating a data model which is not confined to a proprietary language.

#### **BRIEF SUMMARY OF THE INVENTION**

The present invention provides a mechanism for a task oriented data model for a development tool in an object oriented programming system.

According one aspect of the invention, the meta-data of the data model is represented using an open standard. Advantageously, by representing the meta-data using an open standard, conventional parsers for the open standard may be used to retrieve the meta-data. According to this aspect of the present invention, the data model takes the form of a document file expressed in Extensible Markup Language or XML. XML comprises a simplified subset of the Standardized Generalized Markup Language or SGML. Like SGML, but unlike other subsets of SGML such as Hypertext Markup Language (HTML), XML permits (i) users to add new defined tags to the tag set at will; and (ii) XML documents to be properly nested within one another to an arbitrary extent. In accordance with this aspect of the invention, the import and export utility in the object builder is implemented to express the data model as an XML document wherein the data elements are arranged according to a meta data model. According to this aspect of the invention, the meta data model comprises a Document Type Definition or DTD in XML.

According to another aspect of the invention, the data model incorporates a task oriented structure which mirrors the user interface in the object builder. The task tree for the user interface is represented using containment relationships defined by XML constructs specified in the Document Type Definition, i.e. the meta data model. For example, the properties and values in the SmartGuides™ under the Component Broker™ toolkit are represented according to this aspect of the invention as a tag with an identical label and value format. Advantageously, this task oriented data structure allows users to quickly locate and recognize the data element of interest in the data model. In addition, the task oriented structure of the data model provides



the capability to export and import data elements with varying degrees of granularity.

According to an aspect of the invention, there is provided a task oriented data structure embodied in a program storage device for an object oriented application program, the object oriented application program having a user interface comprising a plurality of items, said items defining a sequence of tasks for creating one or more objects, said data structure comprising: (a) a data model expressed in the form of a document according to a meta data language; (b) said data model including a plurality of data elements; (c) each of said data elements corresponding to one of the tasks in said sequence of tasks; and (d) said data elements being arranged according to a meta data model, wherein said meta data model mirrors the sequence of tasks defined in the user interface. Said meta data model may means for validating each of said data elements and the arrangement of said data elements. Further, said meta data language may comprise XML or Extensible Markup Language, and said data elements being arranged according to containment constructs specified in said meta data model. Said means for validating may also comprise a Document Type Definition specified in XML. And, said containment constructs may comprise tag definitions according to XML.

There is also provided, in an application program for creating objects, the application program having a user interface comprising a plurality of items, said items defining a sequence of tasks for creating the objects, and said application program including an export utility for exporting document files, a export utility comprising: (a) means for exporting a document file expressed in a meta data programming language; and (b) wherein said document file comprises a plurality of data elements, each of said data elements

corresponding to one of the tasks in said sequence of tasks, and said data elements being arranged according to a meta data model, wherein said meta data model mirrors the sequence of tasks defined in the user interface. Said meta data model may also include means for validating each of the data elements included in said document file. Said document file may also be expressed in XML or Extensible Markup Language, and said data elements being arranged according to containment constructs specified in said meta data file. And, said means for validating may comprise a Document Type Definition specified in XML.

According to another aspect of the invention, there is provided a computer program product for an object oriented application program for creating objects, the application program having a user interface comprising a plurality of items, said items defining a sequence of tasks for creating the objects, and said application program including an export utility for exporting document files, said computer program product comprising a program storage device; means recorded on said program storage device for instructing a computer to perform the steps of, (a) outputting a document file, said document file being expressed according to a meta data programming language; and (b) said document file comprising a plurality of data elements, each of said data elements corresponding to one of the tasks in said sequence of tasks, and said data elements being arranged according to a meta data model, wherein said meta data model mirrors the sequence of tasks defined in the user interface. Said meta data model may include means for validating each of the data elements in said document file. Said meta data language may comprise XML or Extensible Markup Language, and said data elements being arranged according to containment constructs specified in said meta data model. And, said means for

validating may comprise a Document Type Definition specified in XML.

There is also provided a computer system for creating objects in an object oriented application program, the application program having a user interface comprising a plurality of items, said items defining a sequence of tasks for creating the objects, and said application program including an export utility for exporting document files, said computer system comprising: (a) means for outputting a document file, said document file being expressed according to a meta data programming language; and (b) said document file comprising a plurality of data elements, each of said data elements corresponding to one of the tasks in said sequence of tasks, and said data elements being arranged according to a meta data model, wherein said meta data model mirrors the sequence of tasks defined in the user interface. Said meta data model may include means for validating each of the data elements in said document file. Said meta data language may also comprise XML or Extensible Markup Language, and said data elements being arranged according to containment constructs specified in said meta data model. And, said means for validating may comprise a Document Type Definition specified in XML.

#### BRIEF DESCRIPTION OF THE DRAWINGS

Reference will now be made to the accompanying drawings which show, by way of example, preferred embodiments of the present invention, and in which:

Fig. 1 is a diagrammatic representation of the main window of an object builder of the type suitable for a task oriented data model according to the present invention; and

Fig. 2 shows the main window of the Component Broker object builder with an exemplary display of a user-defined interface object which has been selected.

5 DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The present invention comprises a task oriented structure for a data model suitable for object oriented programming. According to the invention, the task oriented data structure is implemented using the open standard language XML (Extensible Markup Language).  
10 The data model for an object is expressed as an XML document which is defined according to a meta data model. According to the invention, the meta data model is implemented as a Document Type Definition or DTD in XML. The XML document is compatible with the export and import utilities in development tools (e.g. object  
15 builder tool). It is a feature of the present invention that the task oriented structure mirrors the user interface in the object builder or development tool. In the following description, the task oriented data structure is described in the context of an object builder application which is in the "toolkit" for IBM's Component  
20 Broker software package.

Reference is first made to Fig. 1, which shows in diagrammatic form an object builder tool 10 in the Component Broker toolkit. The object builder tool 10 is an application program which provides an integrated development environment for software developers. As  
25 shown in Fig. 1, the object builder tool 10 comprises a main window 20 having four panes denoted by 21, 22, 23, and 24 respectively. The main view pane 21 on the left side of the window 20 provides the main view which is a task tree view. The main view pane 21 is

also the context for the content displayed in the three other panes 22, 23, and 24.

As shown in Fig. 1, the main view pane 21 comprises a "Framework Interfaces" item (or element) 110, a "User-Defined Business Objects" item 120, a "User-Defined Compositions" item 130, a "User-Defined Data Objects" item 140, a "DBA-Defined Schemas" item 150, a "User-Defined PA Schemas" item 160, a "Non-IDL Type Objects" item 170, a "Build Configuration" item 180, an "Application Configuration" item 190, a "Container Definition" item 200, a "Default Homes" item 210, and a "FlowMark" item 220.

The content of the other three panes 22, 23 and 24 in the window 10 depends on the selected item or element in the main view pane 21. Referring to Fig. 2, an item from "User-Defined Business Objects" item 120, specifically "Account", has been selected. In turn, the second pane 22 displays an inheritance graph for the Account item, and the third pane 23 shows the methods and attributes for the Account item. When one of the methods appearing in the third pane 23 is selected, the implementation of that method is shown in the fourth pane 24.

It will be appreciated that the object builder tool 10 utilizes a task tree structure. As shown in Fig. 1, the "User-Defined Business Objects" item 120 has a user interface which is organized as a task tree structure. The task-oriented structure of the user interface forces a user to follow a specified sequence in order to create a business object, i.e. an object containing business methods (logic) and data (state) that is intended for use within business applications. As will now be described, the data model according to the present invention comprises a task oriented structure which mirrors the user interface. The data model includes

all the persistent meta-data which is used by the object builder tool 10.

Referring to Fig. 1, and specifically the main view pane 21, the task tree structure for the "User-Defined Business Objects" item 120 comprises a file item or element 121, a module item 122, an interface item 123, a key helper item 124, a copy helper item 125, a business object implementation item 126, and a data object item 127. The "User-Defined Data Objects" item 140 is also organized in a task tree structure and comprises a file item 141, a module item 142, and a interface item 142. Similar task tree structures are provided for the "DBA-Defined Schemas" item 150, the "Non-IDL Types Objects" item 170, the "Build Configuration" item 180, the "Application Configuration" item 190, and the "Container Definition" item 200.

According to the task tree structure for the "User-Defined Business Objects" item 120, a business object is created by the following sequence of steps: (1) create file; (2) create module; (3) create interface; (4) create key helper; (5) create copy helper; and (6) create implementation.

The first step performed by a user involves creating a file. The object builder 10 generates the file as an IDL (Interface Definition File). As will be understood by one skilled in the art, IDL is a contractual, neutral and declarative language that specifies an object's boundaries and its interfaces, e.g. the aspects of the object that are accessible to the public. IDL provides operating system and programming language independent interfaces to all services and components that reside on a bus as defined according to the CORBA (Common Object Request Broker Architecture) standard. The second step is optional and allows a

user to define a module which comprises a grouping of interfaces. The third step performed by the user involves defining the public interface for the user-defined business object. The fourth step involves creating a key helper which comprises an object which provides a copy of the "identity" of the business object. The fifth step involves creating an object which duplicates everything about the business object, not just its identity. The sixth step comprises implementing the specific behaviour, i.e. methods, for the business object.

Similarly, according to the task tree structure for the "User-Defined Data Objects" item 140, a data object is created according to the following sequence of steps: (1) create file; (2) create module; and (3) create interface. A data object is an object that provides an object-oriented rendering of the application data.

According to the present invention, the data model comprises a task oriented structure which preserves the sequence of steps followed by a user during the creation of the object. The data content of the object, i.e. instance of data, is expressed by the object builder tool in the form of a text document. Since the data model mirrors the exact order in which the object was created by the user, the data model provides a structure from which the user interface of the object builder tool can be inferred. Because the user interface is implicit in the organization of the data model, the data model can be imported by another tool by simply following or scripting the data structure.

According to another aspect of the invention, the task oriented data model is implemented using the meta-data language XML (i.e. Extensible Markup Language) and a meta data model is implemented in the form of a Document Type Definition or DTD which

is also expressed in XML. XML is a meta-data language which is self-descriptive of both services and information. In the context of the present invention, a meta data implementation is preferred because the data can be generically validated by reading or importing the data. The DTD (i.e. meta data model) is used for validating the correctness of the data model which is expressed in the form of an XML document.

According to this aspect of the present invention, the meta data model for the "User-Defined Business Objects" item 120 is specified in form of a Document Type Definition in XML as shown below.

---

```
<?xml encoding="US-ASCII"?>
```

```
<!ELEMENT UserDefinedBusinessObject (BusinessObjectUnit)*>
```

```
<!ELEMENT UserDefinedDataObject (DOFile)*>
```

```
<!ELEMENT BusinessObjectUnit (BOFile | CompositeFile)*>
```

```
<!ELEMENT DBADefinedSchemas (SchemaGroup)*>
```

```
<!ELEMENT UserDefinedPAOs (Schema)*>
```

```
<!-- ***** BO File ***** -->
```

```
<!ELEMENT BOFile (Comments| IncludeFile| Module | BOArtifact |
Constant | Enumeration | Exception | Structure | Typedef | Union | Uuid)*>
<!ATTLIST BOFile Name CDATA #IMPLIED>
```

```
<!ELEMENT Comments (#PCDATA)*>
```

```
<!ELEMENT IncludeFile (#PCDATA)*>
```

```
<!ELEMENT Uuid (#PCDATA)*>
```

```
<!--***** Composite File ***** -->
```

```
<!ELEMENT CompositeFile (Comments| IncludeFile| Module | BOArtifact |
Constant | Enumeration | Exception | Structure | Typedef | Union | Uuid)*>
<!ATTLIST CompositeFile Name CDATA #IMPLIED>
```



```

<!-- ***** Module ***** -->
<!ELEMENT Module (#PCDATA | BOArtifact | DOArtifact | Comments | Constant |
Enumeration | Exception | Structure | Typedef | Union | Uuid)*>
<!ATTLIST Module Name CDATA #IMPLIED>

5
<!-- ***** DO File ***** -->
<!ELEMENT DOFile (Comments | IncludeFile | Module | DOArtifact |
Constant | Enumeration | Exception | Structure | Typedef | Union | Uuid)*>
<!ATTLIST DOFile
10   Name CDATA #IMPLIED>

<!-- ***** BOArtifact ***** -->
<!ELEMENT BOArtifact (Interface | KeyArtifact | CopyArtifact | BOImplementation)*>
<!ELEMENT Interface (#PCDATA | IsQueryable | IsWLM | Comments |
15 InterfaceInheritance | Attribute | Method | Constant | Enumeration |
Exception | Structure | Typedef | Union | Uuid | ObjectRelationship |
CompositeInfo)*>
<!ATTLIST Interface
   Name CDATA #IMPLIED>
20 <!ELEMENT IsQueryable (#PCDATA)*>
<!ELEMENT IsWLM (#PCDATA)*>
<!ELEMENT InterfaceInheritance (#PCDATA)*>

<!-- ***** CompositeInfo ***** -->
25 <!ELEMENT CompositeInfo (CompositeRules | InterfaceReference)*>
<!ATTLIST CompositeInfo
   CompositeToolMoniker CDATA #IMPLIED>

<!-- ***** CompositeRules ***** -->
30 <!ELEMENT CompositeRules (#PCDATA)*>

<!-- ***** InterfaceReference ***** -->
<!ELEMENT InterfaceReference EMPTY>
<!ATTLIST InterfaceReference

```

```

Referenced CDATA #IMPLIED
InstanceInterface CDATA #IMPLIED
InstanceName CDATA #IMPLIED
InstanceKind CDATA #IMPLIED>

```

5

```
<!--***** DataReference ***** -->
```

```
<!ELEMENT DataReference (Prereq | DataStore | Chain | Select)*>
```

```
<!ATTLIST DataReference
```

```
    Type CDATA #IMPLIED
```

10

```
    ReferencedCategory CDATA #IMPLIED
```

```
    ReferencedInterface CDATA #IMPLIED
```

```
    ReferencedName CDATA #IMPLIED
```

```
    ReferencedKind CDATA #IMPLIED
```

```
    InstanceInterface CDATA #IMPLIED
```

15

```
    InstanceName CDATA #IMPLIED
```

```
    InstanceKind CDATA #IMPLIED>
```

```
<!-- ***** Prereq ***** -->
```

```
<!ELEMENT Prereq (Prereq | DataStore | Chain | Select)*>
```

20

```
<!ATTLIST Prereq
```

```
    ReferencedParameterName CDATA #IMPLIED
```

```
    ReferencedParameterKind CDATA #IMPLIED
```

```
    TargetInterface CDATA #IMPLIED
```

```
    OperationName CDATA #IMPLIED
```

25

```
    OperationKind CDATA #IMPLIED
```

```
    ParameterName CDATA #IMPLIED
```

```
    ParameterKind CDATA #IMPLIED
```

```
    Helper CDATA #IMPLIED
```

```
    ReferencedCategory CDATA #IMPLIED
```

30

```
    ReferencedInterface CDATA #IMPLIED
```

```
    ReferencedName CDATA #IMPLIED
```

```
    ReferencedKind CDATA #IMPLIED
```

```
    InstanceInterface CDATA #IMPLIED
```

```
    InstanceName CDATA #IMPLIED>
```

InstanceKind CDATA #IMPLIED>

<!--\*\*\*\*\* DataStore \*\*\*\*\* -->

<!ELEMENT DataStore (#PCDATA)\*>

5 <!ATTLIST DataStore

ReferencedCategory CDATA #IMPLIED

ReferencedInterface CDATA #IMPLIED

ReferencedName CDATA #IMPLIED

ReferencedKind CDATA #IMPLIED

10 InstanceInterface CDATA #IMPLIED

InstanceName CDATA #IMPLIED

InstanceKind CDATA #IMPLIED>

<!-- \*\*\*\*\* Chain \*\*\*\*\* -->

15 <!ELEMENT Chain EMPTY>

<!-- \*\*\*\*\* Select \*\*\*\*\* -->

<!ELEMENT Select (NullAttribute)\*>

20 <!--\*\*\*\*\* NullAttribute \*\*\*\*\* -->

<!ELEMENT NullAttribute (Prereq | DataStore | Chain | Select)\*>

<!ATTLIST NullAttribute

AttributeInterface CDATA #IMPLIED

AttributeName CDATA #IMPLIED

25 AttributeKind CDATA #IMPLIED

Negated CDATA #IMPLIED

ReferencedCategory CDATA #IMPLIED

ReferencedInterface CDATA #IMPLIED

ReferencedName CDATA #IMPLIED

30 ReferencedKind CDATA #IMPLIED

InstanceInterface CDATA #IMPLIED

InstanceName CDATA #IMPLIED

InstanceKind CDATA #IMPLIED>

```

<!--***** DelegateToGroup ***** -->
<!ELEMENT DelegateToGroup (#PCDATA)*>

<!--***** KeyAttrMapGroup ***** -->
5 <!ELEMENT KeyAttrMapGroup (KeyAttrMapInfo)*>

<!--***** KeyAttrMapInfo ***** -->
<!ELEMENT KeyAttrMapInfo (KeyAttrReference)*>
<!ATTLIST KeyAttrMapInfo
10     Interface CDATA #IMPLIED
     AttributeName CDATA #IMPLIED
     AttributeKind CDATA #IMPLIED>

<!--***** KeyAttrReference ***** -->
15 <!ELEMENT KeyAttrReference (KeyAttributeReference |
     InstanceAttributeReference)*>

<!--***** KeyAttributeReference ***** -->
<!ELEMENT KeyAttributeReference EMPTY>
20 <!ATTLIST KeyAttributeReference
     Interface CDATA #IMPLIED
     AttributeName CDATA #IMPLIED
     AttributeKind CDATA #IMPLIED>

<!--***** InstanceAttributeReference ***** -->
25 <!ELEMENT InstanceAttributeReference EMPTY>
<!ATTLIST InstanceAttributeReference
     Interface CDATA #IMPLIED
     AttributeName CDATA #IMPLIED
30     AttributeKind CDATA #IMPLIED>

<!-- ***** LocationGroup ***** -->
<!ELEMENT LocationGroup (LocationByKey)*>

```

```
<!--***** LocationByKey ***** -->
```

```
<!--ELEMENT LocationByKey (InterfaceReference)*>
```

```
<!--ATTLIST LocationByKey
```

```
Group CDATA #IMPLIED
```

```
5 FindObject CDATA #IMPLIED
```

```
CreateObject CDATA #IMPLIED
```

```
RemoveObject CDATA #IMPLIED
```

```
CreateFromCopy CDATA #IMPLIED
```

```
HomeName CDATA #IMPLIED
```

```
10 FactoryFinderName CDATA #IMPLIED
```

```
HomeParameter CDATA #IMPLIED>
```

```
<!--***** Object Relationship ***** -->
```

```
<!--ELEMENT ObjectRelationship (Uuid | ObjectType |
15 ReferenceCollectionImplementation | ReferenceResolvedByForeignKey |
HomeToQuery | Attribute | Method)*>
```

```
<!--ATTLIST ObjectRelationship Name CDATA #IMPLIED>
```

```
<!--ELEMENT ObjectType (#PCDATA)*>
```

```
<!--ELEMENT ReferenceCollectionImplementation (#PCDATA)*>
```

```
20 <!--ELEMENT ReferenceResolvedByForeignKey (#PCDATA)*>
```

```
<!-- ***** Construct ***** -->
```

```
<!--ELEMENT Constant (Type | Initializer | Size | Uuid)*>
```

```
<!--ATTLIST Constant
```

```
25 Name CDATA #IMPLIED>
```

```
<!--ELEMENT Enumeration (Member | Uuid)*>
```

```
<!--ATTLIST Enumeration
```

```
Name CDATA #IMPLIED>
```

```
30
```

```
<!--ELEMENT Exception (Member | Uuid)*>
```

```
<!--ATTLIST Exception Name CDATA #IMPLIED>
```

```

<!ELEMENT Member (Type | Size | TypeSpecification | SequenceSize | ArraySize |
CaseLabel | Uuid)*>
<!ATTLIST Member Name CDATA #IMPLIED>
<!ELEMENT TypeSpecification (#PCDATA)*>
5 <!ELEMENT SequenceSize (#PCDATA)*>
<!ELEMENT ArraySize (#PCDATA)*>
<!ELEMENT CaseLabel (#PCDATA)*>
<!ELEMENT Structure (Member | Uuid)*>
<!ATTLIST Structure Name CDATA #IMPLIED>
10 <!ELEMENT Typedef (Type | Size | TypeSpecification | SequenceSize | ArraySize |
Uuid)*>
<!ATTLIST Typedef Name CDATA #IMPLIED>
<!ELEMENT Union (SwitchSpecifier | Member | Uuid)*>
<!ATTLIST Union Name CDATA #IMPLIED>
15 <!ELEMENT SwitchSpecifier (#PCDATA)*>

<!-- ***** Attribute ***** -->
<!ELEMENT Attribute (Type| Initializer| Size| Implementation| Read-only|
Overridable| (StringBehaviour?) | GetterMethodBodyForPlatform |
20 SetterMethodBodyForPlatform | GetterImplementationType |
SetterImplementationType| ParentInterface | DOAttributeName | Uuid |
DataReference | DelegateToGroup| UsedInImplementation | StaticData |
GetterBodiesKeepAllPlatformsInSyncCpp |
GetterBodiesKeepAllPlatformsInSyncJava |
25 SetterBodiesKeepAllPlatformsInSyncCpp |
SetterBodiesKeepAllPlatformsInSyncJava)*>
<!ATTLIST Attribute
Name CDATA #IMPLIED>
<!ELEMENT Type (#PCDATA)>
30 <!ELEMENT Initializer (#PCDATA)*>
<!ELEMENT Size (#PCDATA)*>
<!ELEMENT Implementation (#PCDATA)>
<!ELEMENT Read-only (#PCDATA)>
<!ELEMENT Overridable (#PCDATA)>

```

```

<!ELEMENT StringBehaviour (#PCDATA)>
<!ELEMENT GetterImplementationType (#PCDATA)*>
<!ELEMENT SetterImplementationType (#PCDATA)*>
<!ELEMENT DOAttributeName (#PCDATA)*>
5  <!ELEMENT ParentInterface (#PCDATA)*>
<!ELEMENT UsedInImplementation (#PCDATA)*>
<!ELEMENT StaticData (#PCDATA)*>
<!ELEMENT GetterBodiesKeepAllPlatformsInSyncCpp (#PCDATA)*>
<!ELEMENT GetterBodiesKeepAllPlatformsInSyncJava (#PCDATA)*>
10 <!ELEMENT SetterBodiesKeepAllPlatformsInSyncCpp (#PCDATA)*>
<!ELEMENT SetterBodiesKeepAllPlatformsInSyncJava (#PCDATA)*>
<!ELEMENT GetterMethodBodyForPlatform (Language)*>
<!ATTLIST GetterMethodBodyForPlatform Name CDATA #IMPLIED>
<!ELEMENT SetterMethodBodyForPlatform (Language)*>
15 <!ATTLIST SetterMethodBodyForPlatform Name CDATA #IMPLIED>
<!--ELEMENT ImplementationLanguage ANY-->
<!ELEMENT Language (UseToolDef, UseFile, UseTDE,Filename?, ToolDef?,
UserDef?,((OOSQLSourceType?, OOSQLUserDef?) | (ESQLSourceType?, ESQLUserDef?)))>
<!ATTLIST Language
20   Name CDATA #IMPLIED>
<!ELEMENT UseToolDef (#PCDATA)>
<!ELEMENT UseFile (#PCDATA)>
<!ELEMENT UseTDE (#PCDATA)>
<!ELEMENT Filename (#PCDATA)*>
25 <!ELEMENT ToolDef (#PCDATA)*>
<!ELEMENT UserDef (#PCDATA)*>
<!ELEMENT OOSQLSourceType (#PCDATA)*>
<!ELEMENT OOSQLUserDef (#PCDATA)*>
<!ELEMENT ESQLSourceType (#PCDATA)*>
30 <!ELEMENT ESQLUserDef (#PCDATA)*>

<!--***** Method ***** -->
<!-- Preserve spaces and newline -->
<!ATTLIST Filename xml:space (default|preserve) "preserve">

```

```

<!-- ATTLIST ToolDef xml:space (default|preserve) "preserve">
<!-- ATTLIST UserDef xml:space (default|preserve) "preserve">
<!-- ATTLIST Comments xml:space (default|preserve) "preserve">
<!-- ATTLIST OOSQLUserDef xml:space (default|preserve) "preserve">
5 <!-- ATTLIST ESQLErrorDef xml:space (default|preserve) "preserve">
<!-- Preserve spaces and newline -->
<!-- ELEMENT Method (ReturnType | Size | Implementation | MethodBodyForPlatform |
One-way | Overridable | ConstMethod | IsFrameworkMethod | ExceptionRaises |
Context | Parameter | Uuid | ParentInterface | DataReference | DelegateToGroup
10 | MethodType | UsedInImplementation | StaticMethod | PushDownMethod |
MappedMethod | PlatformSet | MethodBodiesKeepAllPlatformsInSyncCpp |
MethodBodiesKeepAllPlatformsInSyncJava)*>
<!-- ATTLIST Method
Name CDATA #IMPLIED>
15 <!-- ELEMENT ReturnType (#PCDATA)>
<!-- ELEMENT ConstMethod (#PCDATA)>
<!-- ELEMENT PushDownMethod (#PCDATA)*>
<!-- ELEMENT MappedMethod (#PCDATA)*>
<!-- ELEMENT MethodBodyForPlatform (Language)*>
20 <!-- ATTLIST MethodBodyForPlatform Name CDATA #IMPLIED>
<!-- ELEMENT ImplementationType (#PCDATA)*>
<!-- ELEMENT Designpattern (#PCDATA)*>
<!-- ELEMENT UseMacro (#PCDATA)>
<!-- ELEMENT One-way (#PCDATA)>
25 <!-- ELEMENT IsFrameworkMethod (#PCDATA)>
<!-- ELEMENT ExceptionRaises (#PCDATA)*>
<!-- ELEMENT Context (#PCDATA)*>
<!-- ELEMENT StaticMethod (#PCDATA)*>
<!-- ELEMENT PlatformSet (#PCDATA)>
30 <!-- ELEMENT MethodBodiesKeepAllPlatformsInSyncCpp (#PCDATA)*>
<!-- ELEMENT MethodBodiesKeepAllPlatformsInSyncJava (#PCDATA)*>
<!-- ELEMENT Parameter (ParameterType | Size | DirectionalAttr | Uuid)*>
<!-- ATTLIST Parameter
Name CDATA #IMPLIED>

```



```

<!ELEMENT ParameterType (#PCDATA)*>
<!ELEMENT DirectionalAttr (#PCDATA)*>
<!ELEMENT MethodType (#PCDATA)*>

5  <!--***** DOArtifact ***** -->
    <!ELEMENT DOArtifact (DOInterface| Comments)*>

    <!--***** Key ***** -->
10  <!ELEMENT KeyArtifact (#PCDATA| File | Module | Interface | Attribute | Method
    | InterfaceInheritance | Uuid | Prologue | CppPrologue | JavaPrologue | Epilogue
    | CppEpilogue | JavaEpilogue | KeyAttrMapGroup)*>
    <!ELEMENT File (#PCDATA | Uuid | IncludeFile | Comments )*>
    <!ATTLIST File Name CDATA #IMPLIED>

15  <!--***** Copy ***** -->
    <!ELEMENT CopyArtifact (#PCDATA | File | Module | Interface | Attribute |
    Method | InterfaceInheritance | Uuid | Prologue | CppPrologue |
    JavaPrologue| Epilogue | CppEpilogue | JavaEpilogue)*>

20  <!--***** BO Implementation ***** -->
    <!ELEMENT BOImplementation (#PCDATA| File | Module | Interface | Pattern |
    LazyEvaluation | DataObjectInterface | SessionableBO | ImplementationLanguage |
    UserData | ImplementationInheritance | Attribute | Method | Prologue |
25  CppPrologue | JavaPrologue | Epilogue | CppEpilogue | JavaEpilogue | Key | Copy
    | Handle | DOInterface | MO | Uuid | LocationGroup | ObjectRelationship)*>
    <!ELEMENT Pattern (#PCDATA)>
    <!ELEMENT LazyEvaluation (#PCDATA)>
    <!ELEMENT DataObjectInterface (#PCDATA)>
30  <!ELEMENT SessionableBO (#PCDATA)>
    <!ELEMENT ImplementationLanguage (#PCDATA)*>
    <!ELEMENT UserData (#PCDATA)*>
    <!ELEMENT ImplementationInheritance (#PCDATA)>
    <!ELEMENT Prologue (#PCDATA)*>

```

```

<!--ELEMENT CppPrologue (#PCDATA)*>
<!--ELEMENT JavaPrologue (#PCDATA)*>
<!--ELEMENT Epilogue (#PCDATA)*>
<!--ELEMENT CppEpilogue (#PCDATA)*>
5  <!--ELEMENT JavaEpilogue (#PCDATA)*>
    <!--ELEMENT Key (#PCDATA)*>
    <!--ELEMENT Handle (#PCDATA)*>

<!-- Preserve spaces and newline -->
10 <!--ATTLIST Prologue xml:space (default|preserve) "preserve">
    <!--ATTLIST CppPrologue xml:space (default|preserve) "preserve">
    <!--ATTLIST JavaPrologue xml:space (default|preserve) "preserve">
    <!--ATTLIST Epilogue xml:space (default|preserve) "preserve">
    <!--ATTLIST CppEpilogue xml:space (default|preserve) "preserve">
15 <!--ATTLIST JavaEpilogue xml:space (default|preserve) "preserve">

<!--***** DO Stuff ***** -->
<!--ELEMENT DOInterface (File | Module | Interface | Comments |
InterfaceInheritance | DOImplementation | Attribute | Method | Constant |
20 Enumeration | Exception | Structure | Typedef | Union | Uuid)*>
    <!--ATTLIST DOInterface
        Name CDATA #IMPLIED>

<!--***** DO Implementation ***** -->
25 <!--ELEMENT DOImplementation (#PCDATA | File | Module | Interface |
Environment | PersistentBehaviour | DataAccessPattern | PointerHandle |
Key | Copy | ImplementationInheritance | Attribute | Method | Prologue |
CppPrologue | JavaPrologue | Epilogue | CppEpilogue | JavaEpilogue | PO |
POMapping | Uuid)*>
30 <!--ELEMENT Environment (#PCDATA)*>
    <!--ELEMENT PersistentBehaviour (#PCDATA)*>
    <!--ELEMENT DataAccessPattern (#PCDATA)*>
    <!--ELEMENT PointerHandle (#PCDATA)*>
    <!--ELEMENT Copy (#PCDATA)*>

```

```

<!--***** PO Stuff ***** -->
<!ELEMENT PO (Uuid?, ShortName?, ModuleName?, JavaPackage?, JavaPath?,
PAOService?, POAccessType, Prologue?, CppPrologue?, JavaPrologue?, Epilogue?,
CppEpilogue?, JavaEpilogue?, BindFileName, Comments?, POAttribute*, POMethod*,
5 Schema?)>
<!ATTLIST PO
    Name CDATA #IMPLIED>
<!ELEMENT POAccessType (#PCDATA)>
<!ELEMENT BindFileName (#PCDATA)*>
10 <!ELEMENT POAttribute (Uuid?, Type, Size, ColumnName, KeySequenceNumber?, Key?,
    Getter?, Setter?, GetterBodiesKeepAllPlatformsInSyncCpp?,
    SetterBodiesKeepAllPlatformsInSyncCpp?, GetterMethodBodyForPlatform*,
    SetterMethodBodyForPlatform*)>
<!ATTLIST POAttribute Name CDATA #IMPLIED>
15 <!ELEMENT KeySequenceNum (#PCDATA)*>
<!ELEMENT KeySequenceNumber (#PCDATA)*>
<!ELEMENT POMethod (Uuid?, IsFrameworkMethod, MethodFunction?, Implementation,
    Return Type, POParameter*, PlatformSet?, MethodBodiesKeepAllPlatformsInSyncCpp?,
    MethodBodyForPlatform*)>
20 <!ATTLIST POMethod Name CDATA #IMPLIED>
<!ELEMENT POParameter (Type, Size?, ParmAccessType?)>
<!ATTLIST POParameter Name CDATA #IMPLIED>
<!ELEMENT ParmAccessType (#PCDATA)>

25 <!--***** DOImpl->PO Mappings ***** -->
<!ELEMENT POMapping (AttributeMapping*, MethodMapping*)>
<!ELEMENT AttributeMapping (DataObjectAttribute, PatternType?,
    PrimitiveMapping?, ((POAttributeImplementation+, MappingHelper?) | (KeyMapping |
    StructMapping)*))>
30 <!ELEMENT DataObjectAttribute (#PCDATA)>
<!ELEMENT PatternType (#PCDATA)>
<!ELEMENT PrimitiveMapping (POAttributeImplementation+, MappingHelper*)>
<!ELEMENT POAttributeImplementation (POInstance, POInstanceAttribute)>
<!ELEMENT POInstance (#PCDATA)>

```

```

<!ELEMENT POInstanceAttribute (#PCDATA)>
<!ELEMENT MappingHelper (MappingHelperClass, MappingHelperToTarget,
MappingHelperToSource)*>
<!ELEMENT MappingHelperClass (#PCDATA)*>
5 <!ELEMENT MappingHelperToTarget (#PCDATA)*>
<!ELEMENT MappingHelperToSource (#PCDATA)*>
<!ELEMENT KeyMapping (KeyFullName, HomeToQuery, (KeyAttributeMapping)*)>
<!ELEMENT KeyFullName (#PCDATA)>
<!ELEMENT HomeToQuery (#PCDATA)*>
10 <!ELEMENT KeyAttributeMapping (KeyAttribute, (POAttributeImplementation)*,
MappingHelper?)>
<!ELEMENT KeyAttribute (#PCDATA)>
<!ELEMENT StructMapping (StructFullName, (StructAttributeMapping)*)>
<!ELEMENT StructFullName (#PCDATA)>
15 <!ELEMENT StructAttributeMapping (StructAttribute, PrimitiveMapping?, (KeyMapping
| StructMapping)*)>
<!ELEMENT StructAttribute (#PCDATA)>
<!ELEMENT MethodMapping (DataObjectMethod, AlwaysCompletes,
(POMethodImplementation)*)>
20 <!ELEMENT POMethodImplementation (POInstance, POInstanceMethod)>
<!ELEMENT DataObjectMethod (#PCDATA)*>
<!ELEMENT AlwaysCompletes (#PCDATA)*>
<!ELEMENT POInstanceMethod (#PCDATA)>

25 <!--***** Schema Stuff ***** -->
<!ELEMENT Schema (((DatabaseName, UserName?, SchemaName, ContainedBySchemaGroup,
DatabaseType?, SchemaType, PKConstraint?, PKConstraintComment?, Comments?,
SQLFilename, Column*, ((ForeignKey*) |(ViewSelect?, ViewFrom?, ViewWhere?,
ViewOrderBy?, ViewGroupby?, ViewHaving?, IsUserWrittenClause?, ViewSchema+,
30 ViewColumnMapping?)))) |(SchemaName, ShortName, ModuleName, SchemaType,
JavaPackage, JavaPath, JavaClassName, PAOService, Property*, SchemaMethod*)),
PO*)>
<!ATTLIST Schema
Name CDATA #IMPLIED>

```

```

<!ELEMENT ContainedBySchemaGroup (#PCDATA)*>
<!ELEMENT DatabaseName (#PCDATA)>
<!ELEMENT UserName (#PCDATA)*>
<!ELEMENT SchemaName (#PCDATA)>
5  <!ELEMENT ShortName (#PCDATA)>
    <!ELEMENT ModuleName (#PCDATA)>
    <!ELEMENT SchemaType (#PCDATA)>
    <!ELEMENT PKConstraint (#PCDATA)*>
    <!ELEMENT PKConstraintComment (#PCDATA)*>
10  <!ELEMENT SQLFilename (#PCDATA)*>
    <!ELEMENT JavaPackage (#PCDATA)>
    <!ELEMENT JavaPath (#PCDATA)>
    <!ELEMENT JavaClassName (#PCDATA)>
    <!ELEMENT PAOService (#PCDATA)>
15  <!ELEMENT SchemaMethod (Uuid?, IsFrameworkMethod, MethodFunction?,
    Implementation, ReturnType, SchemaParameter*)>
    <!ATTLIST SchemaMethod Name CDATA #IMPLIED>
    <!ELEMENT MethodFunction (#PCDATA)*>
    <!ELEMENT SchemaParameter (Type, Size?)>
20  <!ATTLIST SchemaParameter Name CDATA #IMPLIED>
    <!ELEMENT Column (ColumnName, ColumnType, NotNull, KeySequenceNumber,
    ColumnSequenceNumber, Length, Scale, ForBitData, Comments)>
    <!ATTLIST Column
        Name CDATA #IMPLIED>
25  <!ELEMENT ColumnName (#PCDATA)>
    <!ELEMENT ColumnType (#PCDATA)>
    <!ELEMENT NotNull (#PCDATA)>
    <!ELEMENT ColumnSequenceNumber (#PCDATA)*>
    <!ELEMENT Length (#PCDATA)>
30  <!ELEMENT Scale (#PCDATA)>
    <!ELEMENT ForBitData (#PCDATA)>
    <!ELEMENT Property (TypeString, TypeQualifier, Size, Key?, Getter?, Setter?,
    Uuid?)>
    <!ATTLIST Property

```

```

    Name CDATA #IMPLIED>
<!ELEMENT TypeString (#PCDATA)>
<!ELEMENT TypeQualifier (#PCDATA)>
<!ELEMENT Getter (#PCDATA)>
5  <!ELEMENT Setter (#PCDATA)>
    <!ELEMENT ForeignKey (IsUnnamed, ForeignKeyName?, TargetSchema,
        Comments, FKMapping*)>
    <!ATTLIST ForeignKey Name CDATA #IMPLIED>
    <!ELEMENT ForeignKeyName (#PCDATA)*>
10 <!ELEMENT IsUnnamed (#PCDATA)>
    <!ELEMENT TargetSchema (DatabaseName, Username?, SchemaName,
        ContainedBySchemaGroup)>
    <!ELEMENT FKMapping (OwningAttribute, TargetAttribute)>
    <!ELEMENT OwningAttribute (#PCDATA)>
15 <!ELEMENT TargetAttribute (#PCDATA)>
    <!ELEMENT ViewSelect (#PCDATA)*>
    <!ELEMENT ViewFrom (#PCDATA)*>
    <!ELEMENT ViewWhere (#PCDATA)*>
    <!ELEMENT ViewOrderBy (#PCDATA)*>
20 <!ELEMENT ViewGroupby (#PCDATA)*>
    <!ELEMENT ViewHaving (#PCDATA)*>
    <!ELEMENT IsUserWrittenClause (#PCDATA)*>
    <!ELEMENT ViewSchema (DatabaseName, Username?, SchemaName,
        ContainedBySchemaGroup)>
25 <!ELEMENT ViewColumnMapping (ViewColumn*)>
    <!ELEMENT ViewColumn (TargetSchema, TargetColumn)>
    <!ATTLIST ViewColumn Name CDATA #IMPLIED>
    <!ELEMENT TargetColumn (#PCDATA)*>

30 <!--***** SchemaGroup Stuff ***** -->
    <!ELEMENT SchemaGroup (DatabaseName?, DatabaseType?, DDLFilename?,
        EditGeneratedFile?, Schema*)>
    <!ATTLIST SchemaGroup
        Name CDATA #IMPLIED>

```

```

<!ELEMENT DatabaseType (#PCDATA)*>
<!ELEMENT DDLFilename (#PCDATA)*>
<!ELEMENT EditGeneratedFile (#PCDATA)*>

5  <!--***** MO Stuff ***** -->
    <!ELEMENT MO (File| Module | Interface | ImplementationInheritance |
        MOApplicationAdaptor | Uuid)*>
    <!ELEMENT MOApplicationAdaptor (#PCDATA)*>
    <!-- UDBO ? -->

```

As shown above the meta data model comprises a Document Type Definition (DTD) which specifies the set of required and optional elements, and their attributes, for the data models which are expressed as XML documents. In addition, the DTD specifies the names of the tags and the relationships among elements in the data model, i.e. XML document. It will also be appreciated that the task tree elements for the "User-Defined Business Objects" item 120 are expressed using containment relationships in the Document Type Definition shown above. For example, the file element or item 121 is defined as <!ELEMENT BOFile(Comments|...|Uuid)\*>, and the module item 122 is defined as <!ELEMENT Module (#PCDATA|...Uuid)\*>. Similarly, in the DTD the "attributes" of the object are defined as <!ELEMENT Attribute (Type |Initializer|...)\*>, and the "methods" of the object are defined as <!ELEMENT Method (ReturnType|...)\*>. The specific syntax of the meta data model expressed in the above DTD will be within the understanding of those familiar with XML.

Next, the meta data model according to the present invention for the "Non-IDL Type Objects" item 170 is specified in the form a Document Type Definition as shown below.

---

```
<?xml encoding="US-ASCII"?>
```

```
<!--ELEMENT NonIDLTypeDefinition (NonIDLType)*-->
```

```
5 <!--ELEMENT NonIDLType (ImplementationLanguage | ImplementationForPlatform |
  PackageName | Uuid)*-->
```

```
<!--ATTLIST NonIDLType
  Name CDATA #IMPLIED-->
```

```
<!--ELEMENT ImplementationLanguage (#PCDATA)*-->
```

```
10 <!--ELEMENT Uuid (#PCDATA)*-->
```

```
<!--ELEMENT PackageName (#PCDATA)*-->
```

```
<!--ELEMENT ImplementationForPlatform ( HeaderFile | LibraryName)*-->
```

```
<!--ATTLIST ImplementationForPlatformName CDATA #IMPLIED-->
```

```
<!--ELEMENT HeaderFile (#PCDATA)*-->
```

```
15 <!--ELEMENT LibraryName (#PCDATA)*-->
```

```
<!-- NonIDLType ? -->
```

---

The specific syntax of the meta data model for the "Non-IDL Type Objects" item as expressed in the above DTD will be within the understanding of those familiar with XML.

The meta data model according to the present invention for the "Build Configuration" item 180 is specified in the form of a Document Type Definition as shown below.

---

```
25 <?xml encoding="US-ASCII"?>
```

```
<!--ELEMENT BuildConfiguration (DLLUnit)*-->
```

```
<!--ELEMENT DLLUnit (Uuid | Tier | Description | ApplicablePlatforms |
  MakefileOptionsForPlatform | FilesInDLL | LinkLibInDLL)*-->
```

```
30 <!--ELEMENT MakefileOptionsForPlatform (LibraryName | MakeOptions | IDLCompile |
  JavaCompile | JCBJavaCompile | CPPCompile | LinkOptions)*-->
```

```
<!--ATTLIST DLLUnit Name CDATA #IMPLIED-->
```



```

<!ATTLIST MakefileOptionsForPlatformName CDATA #IMPLIED>
<!ELEMENT Uuid (#PCDATA)*>
<!ELEMENT Tier (#PCDATA)*>
<!ELEMENT ApplicablePlatforms (#PCDATA)*>
5  <!ELEMENT Description (#PCDATA)*>
    <!ELEMENT LibraryName (#PCDATA)*>
    <!ELEMENT MakeOptions (#PCDATA)*>
    <!ELEMENT IDLCompile (#PCDATA)*>
    <!ELEMENT JavaCompile (#PCDATA)*>
10  <!ELEMENT JCBJavaCompile (#PCDATA)*>
    <!ELEMENT CPPCompile (#PCDATA)*>
    <!ELEMENT LinkOptions (#PCDATA)*>
    <!ELEMENT FilesInDLL (File)*>
    <!ELEMENT File (#PCDATA)*>
15  <!ELEMENT LinkLibInDLL (LinkLib)*>
    <!ELEMENT LinkLib (#PCDATA)*>
    <!-- DLL ? -->

```

---

20 The specific syntax of the meta data model for the "Build Configuration" item as shown in the above DTD will be within the understanding of those familiar with XML.

The meta data model according to the present invention for the "Application Configuration" item 190 is specified in the form of a Document Type Definition as shown below.

```

25  <?xml encoding="US-ASCII"?>

    <!ELEMENT ApplicationConfiguration (ApplicationFamilyUnit)*>
    <!ELEMENT ApplicationFamilyUnit      (Description | Version | DiskSpace | Readme
30  | Application | Uuid)*>
    <!ATTLIST ApplicationFamilyUnit
        Name CDATA #IMPLIED>
    <!ELEMENT Description (#PCDATA)*>

```

```

<!--ELEMENT Version (#PCDATA)*>
<!--ELEMENT DiskSpace (#PCDATA)*>
<!--ELEMENT Readme (#PCDATA)*>
<!--ELEMENT Uuid (#PCDATA)*>

```

5

```

<!--***** Application ***** -->
<!--ELEMENT Application ( (Description, Version, JavaVM, InitialState?,
Executable*) | ManagedObject | Uuid)*>
<!--ATTLIST Application
    Name CDATA #IMPLIED>
<!--ELEMENT JavaVM (#PCDATA)*>
<!--ELEMENT InitialState (#PCDATA)*>
<!--ELEMENT Executable (#PCDATA)*>

```

10

15

```

<!--***** Managed Object ***** -->
<!--ELEMENT ManagedObject ( Dll | Key | Copy | DOImpl | ContainerName | Home |
NameOfFactory | NameOfHome | NameContexts | VisibleInCell | VisibleInWorkGroup
| WorkLoadManaged | State | Uuid)*>
<!--ATTLIST ManagedObject Name CDATA #IMPLIED>

```

20

```

<!--ELEMENT Dll (#PCDATA)*>
<!--ELEMENT Key (Dll)>
<!--ATTLIST Key Name CDATA #IMPLIED>
<!--ELEMENT Copy (Dll)*>
<!--ATTLIST Copy Name CDATA #IMPLIED>
<!--ELEMENT DOImpl (Dll)>
<!--ATTLIST DOImpl Name CDATA #IMPLIED>
<!--ELEMENT ContainerName (#PCDATA)>
<!--ELEMENT Home (CustomizedHome | Dll)*>
<!--ATTLIST Home Name CDATA #IMPLIED>

```

25

30

```

<!--ELEMENT CustomizedHome (#PCDATA)>
<!--ELEMENT NameOfFactory (#PCDATA)*>
<!--ELEMENT NameOfHome (#PCDATA)*>
<!--ELEMENT NameContexts (#PCDATA)*>
<!--ELEMENT VisibleInCell (#PCDATA)>
<!--ELEMENT VisibleInWorkGroup (#PCDATA)>
<!--ELEMENT WorkLoadManaged (#PCDATA)>

```

35

```
<!ELEMENT State (#PCDATA)>
```

```
<!-- APPL ? -->
```

---

The specific syntax of the meta data model for the "Application Configuration" item as shown in the above DTD will be within the understanding of those familiar with XML.

The meta data model according to the present invention for the "Container Definition" item 200 is specified in the form of a Document Type Definition as shown below.

---

```
<?xml encoding="US-ASCII"?>
```

```
<!ELEMENT ContainerDefinition (Container)*>
```

```
<!ELEMENT Container (Uuid | Description | NumberOfComponents | TransactionPolicy
| TerminationPolicy | PassivationPolicy | PersistentReferences | BOPattern |
DOPattern | CachingServices | HomeServices | WorkLoadManaged | PolicyGroup |
UsesSession | SessionPolicy | SessionConnectionType | SessionConnectionName |
SessionPriority | PAAService )*>
```

```
<!-- ATTLIST Container Name CDATA #IMPLIED -->
```

```
<!ELEMENT Uuid (#PCDATA)*>
```

```
<!ELEMENT Description (#PCDATA)*>
```

```
<!ELEMENT NumberOfComponents (#PCDATA)*>
```

```
<!ELEMENT TransactionPolicy (#PCDATA)*>
```

```
<!ELEMENT TerminationPolicy (#PCDATA)*>
```

```
<!ELEMENT PassivationPolicy (#PCDATA)*>
```

```
<!ELEMENT PersistentReferences (#PCDATA)*>
```

```
<!ELEMENT BOPattern (#PCDATA)*>
```

```
<!ELEMENT DOPattern (#PCDATA)*>
```

```
<!ELEMENT CachingServices (#PCDATA)*>
```

```
<!ELEMENT HomeServices (#PCDATA)*>
```

```
<!ELEMENT WorkLoadManaged (#PCDATA)*>
```

```
<!ELEMENT PolicyGroup (#PCDATA)*>
```

```
<!ELEMENT UsesSession (#PCDATA)*>
```

```

<!ELEMENT SessionPolicy (#PCDATA)*>
<!ELEMENT SessionConnectionType (#PCDATA)*>
<!ELEMENT SessionPriority (#PCDATA)*>
<!ELEMENT SessionConnectionName (#PCDATA)*>
5 <!ELEMENT PAAService (#PCDATA)*>
  <!-- CONT ? -->

```

10 The specific syntax of the meta data model for the "Container Definition" item as shown in the above DTD will be within the understanding of those familiar with XML.

15 The DTD's shown above provide the meta data models which define the content and arrangement of the data models (i.e. XML documents) generated by the object builder tool 10, and the specified tags provide a mechanism for interpreting the data elements expressed in the XML document. When an XML document is imported by an object builder tool or some other XML compatible tool, the XML document is parsed and rendered according to the specifications in the meta data model, i.e. the DTD. Because the DTD provides a description of the data model, an application can

20 validate that the XML document conforms to its specified structure, i.e. data model. This validation can be automated as the XML language is machine readable.

25 The task oriented data model for the object builder as specified in the DTD's above allows the data to be exported and imported with the following granularity: (1) the entire data model; (2) all business objects under the task tree for the "User-Defined Business Objects" item 120 (Fig. 1); (3) a single Business Object unit; (4) all data objects under the task tree for the "User-Defined Data Objects" item 140 (Fig. 1); (5) a single Data Object

30 unit; (5) all schemas under the task tree for the "DBA-Defined

Schemas" item 150 (Fig. 1); (6) all non-IDL types under the task tree for the "Non-IDL Type Objects" item 170 (Fig. 1); (7) all makefiles under the task tree for the "Build Configuration" item 180 (Fig. 1); (8) all application families under the task tree for the "Application Configuration" item 190 (Fig. 1); (9) all containers under the task tree for the "Container Definition" item 200 (Fig. 1).

Advantageously, the human-readable form of the data model (i.e. the XML document) and the meta data model (i.e. the DTD) allow a user to directly examine the data model and quickly locate the data element or item of interest.

The present invention may be embodied in other specific forms without departing from the spirit or essential characteristics thereof. For example, while the task oriented data model was described in the context of an XML implementation, it will be understood that implementations using other meta data language specifications can be used. Therefore, the presently discussed embodiments are considered to be illustrative and not restrictive, the scope of the invention being indicated by the appended claims rather than the foregoing description, and all changes which come within the meaning and range of equivalency of the claims are therefore intended to be embraced therein.

The invention may be implemented as a computer program product comprising a program storage device and means recorded on said program storage device for instructing a computer to perform the steps of the invention and as a data structure embodied in a program storage device. Such a program storage device may include diskettes, optical discs, tapes, CD-ROMS, hard drives, memory

including ROM or RAM, computer tapes or other storage media capable of storing a computer program.

The invention may also be implemented in a computer system. In a preferred embodiment, a system is provided comprising a computer program operating on a data processing system, with the computer program embodying the method of the invention and producing an output of the method on a display or output device. Data processing systems include computers, computer networks, embedded systems and other systems capable of executing a computer program. A computer includes a processor and a memory device and optionally, a storage device, a video display and/or an input device. Computers may equally be in stand-alone form (such as the traditional desktop personal computer) or integrated into another apparatus (such as a cellular telephone).

While this invention has been described in relation to preferred embodiments, it will be understood by those skilled in the art that changes in the details of processes and structures may be made without departing from the spirit and scope of this invention. Many modifications and variations are possible in light of the above teaching. Thus, it should be understood that the above described embodiments have been provided by way of example rather than as a limitation and that the specification and drawing are, accordingly, to be regarded in an illustrative rather than a restrictive sense.

## CLAIMS

The embodiments of the invention in which an exclusive property or privilege is claimed are defined as follows:

1. A task oriented data structure embodied in a program storage  
5 device for an object oriented application program, the object  
oriented application program having a user interface comprising a  
plurality of items, said items defining a sequence of tasks for  
creating one or more objects, said data structure comprising:

(a) a data model expressed in the form of a document  
10 according to a meta data language;

(b) said data model including a plurality of data elements;

(c) each of said data elements corresponding to one of the  
tasks in said sequence of tasks; and

(d) said data elements being arranged according to a meta  
15 data model, wherein said meta data model mirrors the sequence of  
tasks defined in the user interface.

2. The task oriented data structure embodied in a program storage  
device as claimed in claim 1, wherein said meta data model includes  
20 means for validating each of said data elements and the arrangement  
of said data elements.

3. The task oriented data structure embodied in a program storage  
device as claimed in claim 1 or claim 2, wherein said meta data  
25 language comprises XML or Extensible Markup Language, and said data

elements being arranged according to containment constructs specified in said meta data model.

4. The task oriented data structure embodied in a program storage device as claimed in claim 3, wherein said means for validating comprises a Document Type Definition specified in XML.

5. The task oriented data structure embodied in a program storage device as claimed in claim 3 or claim 4, wherein said containment constructs comprise tag definitions according to XML.

6. In an application program for creating objects, the application program having a user interface comprising a plurality of items, said items defining a sequence of tasks for creating the objects, and said application program including an export utility for exporting document files, said export utility comprising:

(a) means for exporting a document file expressed in a meta data programming language; and

(b) wherein said document file comprises a plurality of data elements, each of said data elements corresponding to one of the tasks in said sequence of tasks, and said data elements being arranged according to a meta data model, wherein said meta data model mirrors the sequence of tasks defined in the user interface.

7. The export utility as claimed in claim 6, wherein said meta data model includes means for validating each of the data elements included in said document file.



8. The export utility as claimed in claim 6 or claim 7, wherein said document file is expressed in XML or Extensible Markup Language, and said data elements being arranged according to containment constructs specified in said meta data file.

5 9. The export utility as claimed in claim 8, wherein said means for validating comprises a Document Type Definition specified in XML.

10 10. A computer program product for an object oriented application program for creating objects, the application program having a user interface comprising a plurality of items, said items defining a sequence of tasks for creating the objects, and said application program including an export utility for exporting document files,  
15 said computer program product comprising:

a program storage device;

means recorded on said program storage device for instructing a computer to perform the steps of,

20 (a) outputting a document file, said document file being expressed according to a meta data programming language; and

(b) said document file comprising a plurality of data elements, each of said data elements corresponding to one of the tasks in said sequence of tasks, and said data elements being arranged according to a meta data model, wherein said meta data  
25 model mirrors the sequence of tasks defined in the user interface

11. The computer program product as claimed in claim 10, wherein said meta data model includes means for validating each of the data elements in said document file.

12. The computer program product claimed in claim 10 or claim 11, wherein said meta data language comprises XML or Extensible Markup Language, and said data elements being arranged according to containment constructs specified in said meta data model.

5

13. The computer program product as claimed in claim 12, wherein said means for validating comprises a Document Type Definition specified in XML.

10

14. A computer system for creating objects in an object oriented application program, the application program having a user interface comprising a plurality of items, said items defining a sequence of tasks for creating the objects, and said application program including an export utility for exporting document files,

15

said computer system comprising:

(a) means for outputting a document file, said document file being expressed according to a meta data programming language; and

20

(b) said document file comprising a plurality of data elements, each of said data elements corresponding to one of the tasks in said sequence of tasks, and said data elements being arranged according to a meta data model, wherein said meta data model mirrors the sequence of tasks defined in the user interface.

25

15. The computer system as claimed in claim 14, wherein said meta data model includes means for validating each of the data elements in said document file.

16. The computer system claimed in claim 14 or claim 15, wherein said meta data language comprises XML or Extensible Markup

Language, and said data elements being arranged according to  
containment constructs specified in said meta data model.

17. The computer system as claimed in claim 16, wherein said means  
5 for validating comprises a Document Type Definition specified in  
XML.

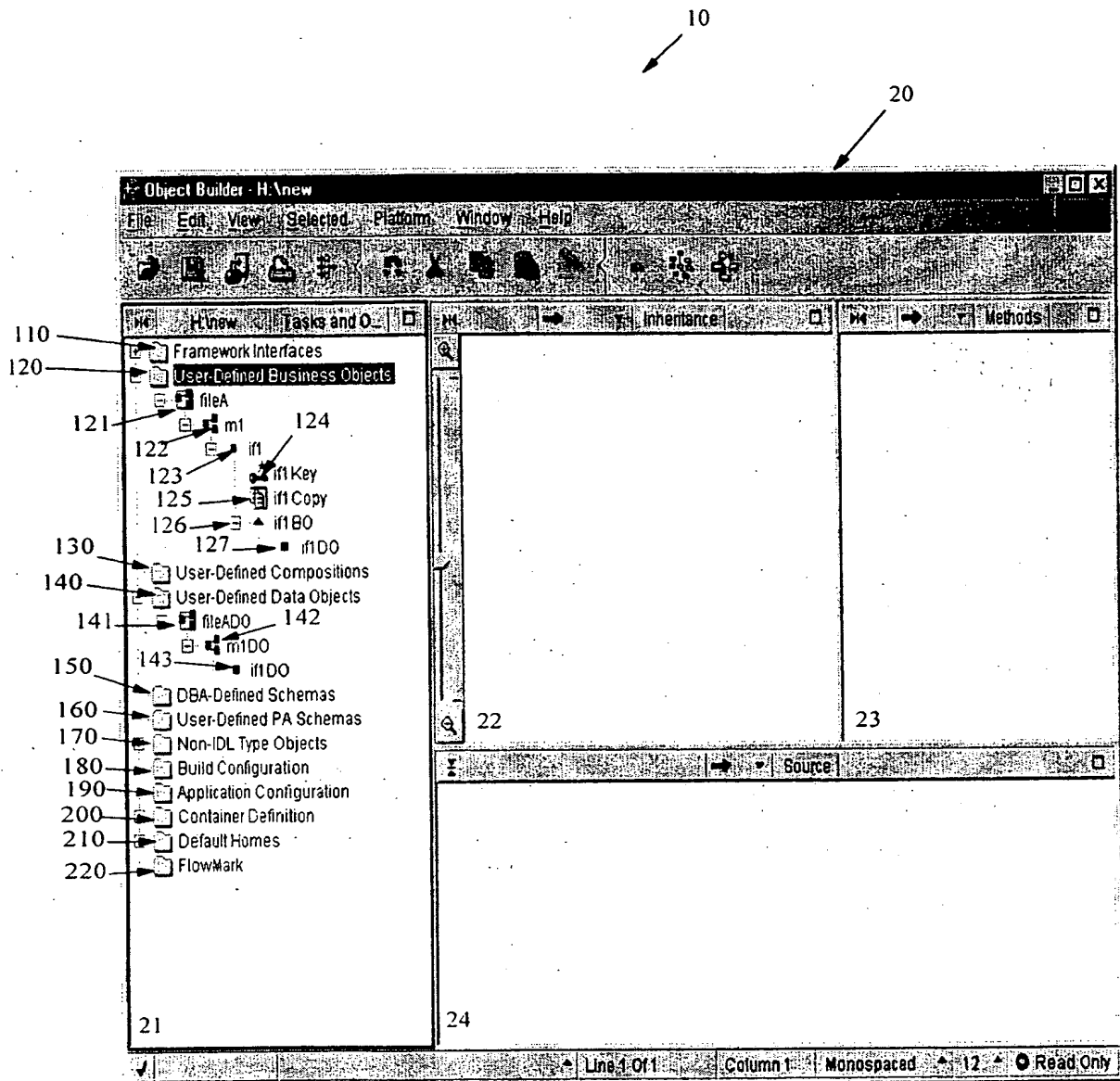


FIGURE 1

10

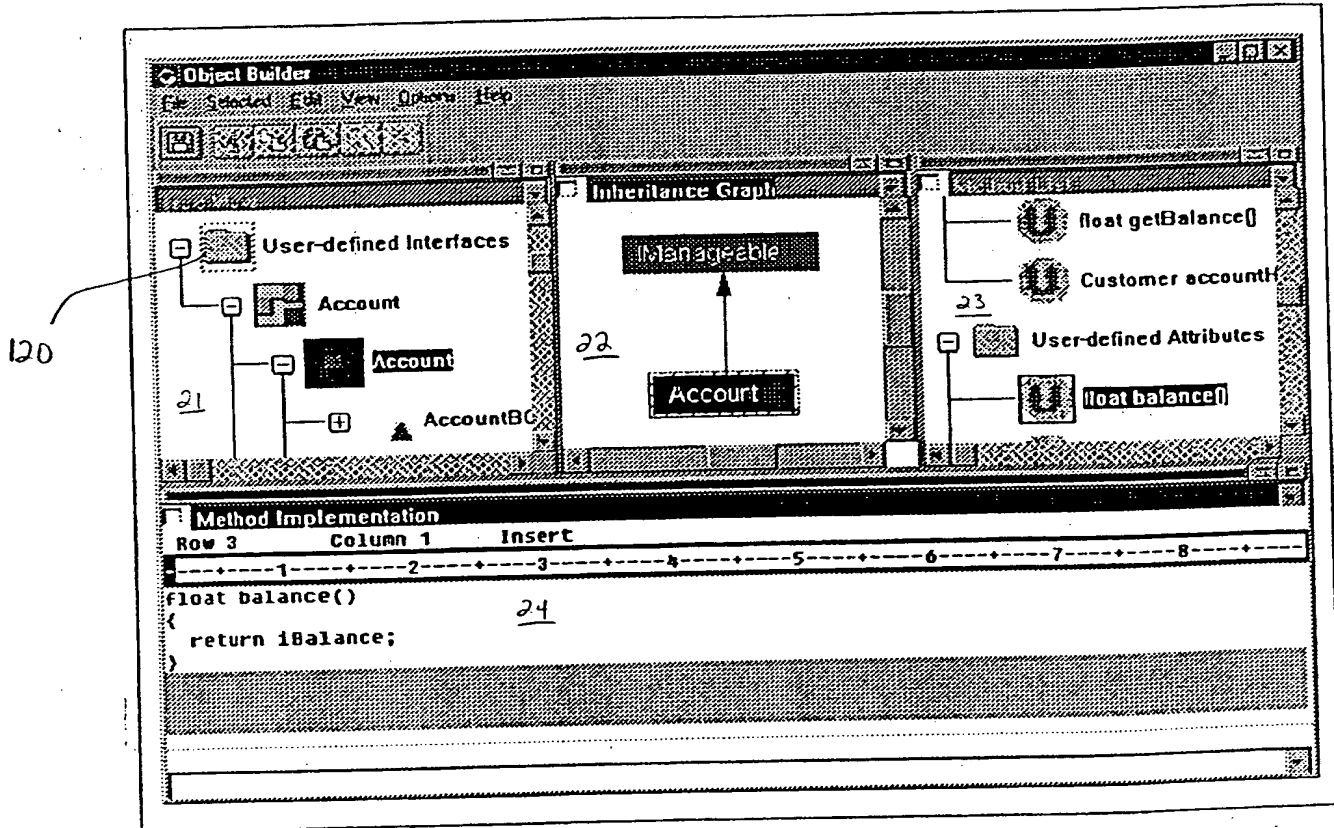


FIG. 2

**THIS PAGE BLANK (USPTO)**